Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 7 — Musterlösung

Aufgabe 20 – 22 (10 + 10 + 20 Punkte)

Der **Assembler-Code** für die Aufgaben 20 – 22:

```
********************************************************************************
********************************************************************************
**
**  Interpreter for the SECDH-2 machine
**
********************************************************************************
********************************************************************************


********************************************************************************
*
* Here, we define the opcodes for the abstract machine.
* Since we want to use them as offsets into an opcode table, they have to be
* aligned to long, i.e., they have to be a multiple of 4.
* The last entry should be LAST_OP which is used to determine the size of the
* opcode table only.
*

PUSH_S          EQU     $00000000
PUSH_ES         EQU     $00000004
MKFRAME         EQU     $00000008
FREE            EQU     $0000000c
BRC             EQU     $00000010
CALL            EQU     $00000014
RET             EQU     $00000018
AP              EQU     $0000001c
PLUS            EQU     $00000020
MINUS           EQU     $00000024
MULT            EQU     $00000028
EQ              EQU     $0000002c
GT              EQU     $00000030
ARGS            EQU     $00000034
LINK            EQU     $00000038
CREATE_CLOS     EQU     $0000003c
DONE            EQU     $00000040
LAST_OP         EQU     $00000044


********************************************************************************

                ORG     $0
                DC.L    $8000           Stack pointer value after a reset
                DC.L    START           Program counter value after a reset


********************************************************************************
*
* Here, we reserve some space for the opcode table
*

OP_TAB          DS.L    LAST_OP


********************************************************************************
*
* Here, we reserve memory space for the stacks S, E, and D.
* The labels x_STACK_T point to the stack bottom:
*

```

```
STACK_REG       DS.L    100
S_STACK_T       DS.L    100
E_STACK_T       DS.L    100
D_STACK_T       DS.L    1


********************************************************************************
*
* In the following part of the memory we keep our programs, i.e., the portion
* of the heap denoted by C
*

P_FAC           DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_S,0,EQ,BRC,P_T,P_F,FREE,1,RET
P_T             DC.L    PUSH_S,1,RET
P_F             DC.L    PUSH_S,1,PUSH_ES,0,0,MINUS,LINK,0,CALL,P_FAC,PUSH_ES,0,0,MULT,RET

P_FAC_PRG       DC.L    PUSH_S,4,LINK,-1,CALL,P_FAC,DONE
* ^^^^^^^ Example


P_Y             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_A,LINK,-1,CALL,P_A,FREE,1,RET
P_A             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_A2,PUSH_ES,1,0,AP,1,FREE,1,RET
P_A2            DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_ES,1,0,PUSH_ES,1,0,AP,1,AP,1,FREE,1,RET

P_FACP          DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_FACP2,FREE,1,RET
P_FACP2         DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_S,0,EQ,BRC,P_TP,P_FP,FREE,1,RET
P_TP            DC.L    PUSH_S,1,RET
P_FP            DC.L    PUSH_S,1,PUSH_ES,0,0,MINUS,PUSH_ES,1,0,AP,1,PUSH_ES,0,0,MULT,RET

P_FACP_PRG      DC.L    PUSH_S,4,CREATE_CLOS,-1,P_FACP,LINK,-1,CALL,P_Y,AP,1,DONE
* ^^^^^^^ Exercise 22d


P_K             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_KI,FREE,1,RET
P_KI            DC.L    ARGS,1,MKFRAME,1,PUSH_ES,1,0,FREE,1,RET

P_AP3           DC.L    ARGS,1,MKFRAME,1,PUSH_S,3,PUSH_ES,0,0,AP,1,FREE,1,RET

P_K_PRG2        DC.L    PUSH_S,2,LINK,-1,CALL,P_K,LINK,-1,CALL,P_AP3,DONE
* ^^^^^^^ Exercise 22c

P_K_PRG         DC.L    PUSH_S,3,PUSH_S,2,LINK,-1,CALL,P_K,AP,1,DONE
* ^^^^^^^ Exercise 22b


P_RC_G          DC.L    ARGS,2,MKFRAME,2,PUSH_ES,1,1,PUSH_ES,0,1,GT,BRC,P_RC_GT,P_RC_GF,FREE,2,RET
P_RC_GT         DC.L    PUSH_ES,0,0,PUSH_S,1,PUSH_ES,1,1,MINUS,LINK,0,CALL,P_RC_G,RET
P_RC_GF         DC.L    PUSH_S,1,PUSH_ES,0,1,PLUS,PUSH_ES,1,0,LINK,1,CALL,P_RC_F,RET
P_RC_F          DC.L    ARGS,2,MKFRAME,2,PUSH_ES,0,1,PUSH_ES,0,0,LINK,-1,CALL,P_RC_G,FREE,2,RET
P_RC_PRG        DC.L    PUSH_S,2,PUSH_S,1,LINK,-1,CALL,P_RC_F,DONE
* ^^^^^^^ Exercise 22a


********************************************************************************
*
* Here, we reserve a heap area for storing the closures:
*

HEAP_START      DS.L    1000
HEAP_STOP       DS.L    1


********************************************************************************
********************************************************************************
*
* Here, the implementation of the SEMCD-H2 interpreter starts:
*

********************************************************************************
*
* First, we define some string constants:
*

START_STR       DC.B    'starting secdh-2 interpreter...',0
```

```
DONE_STR        DC.B    'finished',0,0
RES_STR         DC.B    'the result is: ',0


********************************************************************************
*
* Now, the main program:
*

START:          JSR     INIT_OP_TAB             initialize opcode table
                JSR     INIT_IO                 initialize DUART

                JSR     CLEAR_SCREEN
                MOVE.L  #START_STR,-(SP)
                JSR     PUT_STR

                MOVE.L  #P_FACP_PRG,-(SP)       push program pointer on stack
                JSR     EXEC                    execute program

                MOVE.L  #DONE_STR,-(SP)
                JSR     PUT_STR
                JSR     NEW_LINE
                MOVE.L  #RES_STR,-(SP)
                JSR     PUT_STR
                JSR     PUT_OBJ                 print result (still on stack)
                JSR     NEW_LINE
                BREAK

*  second run -- this time in debugging mode

                JSR     CLEAR_SCREEN
                MOVE.L  #START_STR,-(SP)
                JSR     PUT_STR

                MOVE.L  #P_FACP_PRG,-(SP)       push program pointer on stack
                JSR     EXEC_IO                 execute program

                MOVE.L  #DONE_STR,-(SP)
                JSR     PUT_STR
                JSR     NEW_LINE
                MOVE.L  #RES_STR,-(SP)
                JSR     PUT_STR
                JSR     PUT_OBJ                 print result (still on stack)
                JSR     NEW_LINE
                BREAK


********************************************************************************
*
* function:  void INIT_OP_TAB()
*
* description:
*   initializes the opcode table at OP_TAB with pointers to the subroutines
*   I_xxx that implement the operations xxx.
*
* registers affected: none
*

INIT_OP_TAB:    MOVE.L  A0,-(SP)                save A0
                MOVE.L  #OP_TAB,A0
                MOVE.L  #I_PUSH_S,PUSH_S(A0)
                MOVE.L  #I_PUSH_ES,PUSH_ES(A0)
                MOVE.L  #I_LINK,LINK(A0)
                MOVE.L  #I_MKFRAME,MKFRAME(A0)
                MOVE.L  #I_FREE,FREE(A0)
                MOVE.L  #I_BRC,BRC(A0)
                MOVE.L  #I_CALL,CALL(A0)
                MOVE.L  #I_ARGS,ARGS(A0)
                MOVE.L  #I_RET,RET(A0)
                MOVE.L  #I_AP,AP(A0)
                MOVE.L  #I_CREATE_CLOS,CREATE_CLOS(A0)
                MOVE.L  #I_PLUS,PLUS(A0)
                MOVE.L  #I_MINUS,MINUS(A0)
                MOVE.L  #I_MULT,MULT(A0)
```

```
                MOVE.L  #I_EQ,EQ(A0)
                MOVE.L  #I_GT,GT(A0)
                MOVE.L  #I_DONE,DONE(A0)
                MOVE.L  (SP)+,A0                restore A0
                RTS


*********************************************************************************
*
* function:  (SP1) DEREF( (SP1), (SP2))
*
* description:
*   dereferences the pointer (SP1) (SP2) many times and returns the resulting
*   pointer. It is assumed that (SP2) >= #0 holds and that all pointers to be
*   followed are within the legal range.
*
* registers affected: none
*

DEREF:          LINK    A6,#0
                MOVEM.L D0/A0,-(SP)     save A0 and D0
                MOVE.L  8(A6),D0        fetch sp2
                MOVE.L  12(A6),A0       fetch sp1

                CMPI.L  #0,D0
                BEQ     DREF_DONE
DR_LOOP:        MOVE.L  (A0),A0         deref once
                SUBI.L  #1,D0
                BNE     DR_LOOP
                MOVE.L  A0,12(A6)       write result on SP1

DREF_DONE:      MOVEM.L (SP)+,D0/A0     restore A0 and D0
                UNLK    A6              restore A6
                MOVE.L  (SP),4(SP)      eliminate argument frame
                ADDQ.L  #4,SP
                RTS


*********************************************************************************
*
* function:  (SP1) COPY_ENV( (SP1), (SP2))
*
* description:
*   Expects (SP2) to point to an environment which is t be copied into the
*   memory starting at (SP1). On return, (SP1) points to the memory cell right
*   after the last cell copied.
*
* registers affected: none
*

COPY_ENV:       LINK    A6,#0
                MOVEM.L A0-A2/D0-D1,-(SP)
                MOVE.L  8(A6),A0        fetch sp2 (env ptr)
                MOVE.L  12(A6),A1       fetch sp1 (heap loc)

CE_LOOP:        MOVE.L  (A0)+,A2        store link pointer of env in A2
                CMPA.L  #0,A2           is this the nil-link?
                BEQ     CE_DONE
                MOVE.L  (A0)+,D0        store frame size in D0
                MOVE.L  D0,D1           compute addr of next frame in D1:
                MULU.W  #4,D1             frame-size * 4
                ADD.L   A1,D1             + actual addr + 8 (link / size entry)
                ADDI.L  #8,D1           now D1 holds addr of next heap frame
                MOVE.L  D1,(A1)+        store this ptr in heap
                MOVE.L  D0,(A1)+        store frame size in heap
CE_FR_LOOP:     CMPI.L  #0,D0
                BEQ     CE_FR_DONE
                MOVE.L  (A0)+,(A1)+     copy next entry into heap
                SUBI.L  #1,D0
                BRA     CE_FR_LOOP
CE_FR_DONE:     MOVE.L  A2,A0           make link ptr (in e) the new env ptr
                BRA     CE_LOOP

CE_DONE:        MOVE.L  #0,(A1)+        write terminating NULL frame
```

```
                MOVE.L  A1,12(A6)       write final heap ptr into SP1
                MOVEM.L (SP)+,A0-A2/D0-D1
                UNLK    A6              restore A6
                MOVE.L  (SP),4(SP)      eliminate argument frame
                ADDQ.L  #4,SP
                RTS


****************************************************************************************
*
* function:  (SP) EXEC( (SP))
*
* description:
*   This is the main execution cycle which initializes the SECDH-2 interpreter,
*   and starts the interpretation at (SP).
*   Basically, it repeatedly reads the opcode behind the P_C, looks up the
*   address of the code that implements the operation (in the opcode table), and
*   jumps to that code.
*
*   The last instruction should be "DONE" which leads to the jump address I_DONE
*   and thus exits the loop EXEC_LOOP.
*   The result found on top of P_S is returned via (SP).
*
*   NOTE: A6 serves two purposes here. During the program interpretation, it
*         serves as pointer into the heap P_H; otherwise, it is used as frame
*         pointer.
*
* registers affected: none
*

EXEC:           LINK    A6,#0
                MOVEM.L D0-D7/A0-A5,-(SP)
                MOVE.L  8(A6),A0        fetch SP (program ptr)

                MOVE.L  A6,-(SP)        save frame pointer

                MOVE.L  #S_STACK_T,A2   init P_S (A2)
                MOVE.L  #E_STACK_T,A3   init P_E (A3)
                MOVE.L  #0,-(A3)        push nil-link to P_E
                MOVE.L  A0,A4           init P_C (A4) with prg to be evaluated
                MOVE.L  #D_STACK_T,A5   init P_D (A5)
                MOVE.L  #HEAP_START,A6  init P_H (A6)
                MOVE.L  #OP_TAB,A0

EXEC_LOOP:      MOVE.L  (A4)+,D0        load opcode into D0 and increment P_C
                MOVE.L  (A0,D0.L),A1    dispatch opcode into instruction address
                JSR     (A1)
                CMPA.L  #I_DONE,A1
                BNE     EXEC_LOOP

                MOVE.L  (SP)+,A6        restore frame pointer

                MOVE.L  (A2)+,8(A6)     move result (top of P_S) into SP
                MOVEM.L (SP)+,D0-D7/A0-A5
                UNLK    A6
                RTS


****************************************************************************************
*
* function:  (SP) EXEC_IO( (SP))
*
* description:
*   Same as EXEC together with I/O.
*
* registers affected: none
*

EXEC_IO:        LINK    A6,#0
                MOVEM.L D0-D7/A0-A5,-(SP)
                MOVE.L  8(A6),A0        fetch SP (program ptr)

                MOVE.L  A6,-(SP)        save frame pointer
```

```

                MOVE.L  #S_STACK_T,A2   init P_S (A2)
                MOVE.L  #E_STACK_T,A3   init P_E (A3)
                MOVE.L  #0,-(A3)        push nil-link to P_E
                MOVE.L  A0,A4           init P_C (A4) with prg to be evaluated
                MOVE.L  #D_STACK_T,A5   init P_D (A5)
                MOVE.L  #HEAP_START,A6  init P_H (A6)
                MOVE.L  #OP_TAB,A0

EXEC_IO_LOOP:   MOVE.L  A2,-(SP)         *
                MOVE.L  #S_STACK_T,-(SP) *  push arguments for
                MOVE.L  A3,-(SP)         *     printing actual state
                MOVE.L  A4,-(SP)         *
                MOVE.L  A5,-(SP)         *
                MOVE.L  #D_STACK_T,-(SP) *
                JSR     PR_STATE
                JSR     GETC
                MOVE.L  (A4)+,D0        load opcode into D0 and increment P_C
                MOVE.L  (A0,D0.L),A1    dispatch opcode into instruction address
                JSR     (A1)
                CMPA.L  #I_DONE,A1
                BNE     EXEC_IO_LOOP

                MOVE.L  (SP)+,A6        restore frame pointer

                MOVE.L  (A2)+,8(A6)     move result (top of P_S) into SP
                MOVEM.L (SP)+,D0-D7/A0-A5
                UNLK    A6
                RTS


********************************************************************************
*
* Now, the implementations of the individual operations follow.
*
********************************************************************************
*
* function:  A2, A3, A4, A5, A6 I_xxx( A2, A3, A4, A5, A6)
*
* description:
*   Implements the SECDH-2 operation xxx.
*   A2 contains P_S, A3 contains P_E, A4 contains P_C, A5 contains P_D,
*   A6 contains P_H.
*
* registers which might be affected: D0-D7
*

I_PUSH_S:       MOVE.L  (A4)+,D0        load atom from code heap
                LSL.L   #2,D0
                ORI.L   #1,D0           mark as integer
                MOVE.L  D0,-(A2)        push atom on s
                RTS

I_PUSH_ES:      MOVE.L  A0,-(SP)        save A0
                MOVE.L  (A4)+,D0        fetch parameter i
                MOVE.L  (A4)+,D1        fetch parameter j
                MOVE.L  A3,-(SP)
                MOVE.L  D0,-(SP)
                JSR     DEREF           get i-th indirection starting at P_E
                MOVE.L  (SP)+,A0        fetch i-th indirection
                LSL.L   #2,D1           multiply D1 by 4
                                        * (log. offset => byte offset)
                MOVE.L  8(A0,D1.L),-(A2)  pick j-th entry of the i-th env
                                        * and push it on s
                MOVE.L  (SP)+,A0        restore A0
                RTS

I_MKFRAME:      MOVE.L  (A4)+,D0        fetch frame size parameter
                MOVE.L  (A2)+,D1        fetch link pointer for new env frame
                MOVE.L  D0,D2           preserve frame size in D2
I_MKF_LOOP:     CMPI.L  #0,D0
                BEQ     I_MKF_DONE
                MOVE.L  (A2)+,-(A3)     move top elem from s to e
                SUBI.L  #1,D0
```

```
                BRA     I_MKF_LOOP
I_MKF_DONE:     MOVE.L  D2,-(A3)        push frame size on stack
                MOVE.L  D1,-(A3)        push link pointer on e
                RTS

I_FREE:         MOVE.L  (A4)+,D0        fetch parameter
                ADDI.L  #2,D0           +2 in order to skip link and size info
                MULU.W  #4,D0           multiply by length of entry (4 byte)
                ADD.L   D0,A3
                RTS

I_BRC:          MOVE.L  (A4)+,D0        fetch p_then
                MOVE.L  (A4)+,D1        fetch p_else
                MOVE.L  (A2)+,D2        fetch predicate from s
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                CMPI.L  #7,D2
                BEQ     I_BRC_ELSE
                MOVE.L  D0,A4           set P_C to p_then
                RTS
I_BRC_ELSE:     MOVE.L  D1,A4           set P_C to p_else
                RTS

I_CALL:         MOVE.L  (A4)+,D0        load P_C from program code
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                MOVE.L  D0,A4           set P_C to new address
                RTS

I_RET:          MOVE.L  (A5)+,A4        retore P_C from d
                RTS

I_AP:           MOVEM.L A0-A1,-(SP)
                MOVE.L  (A4)+,D0        fetch parameter r
                MOVE.L  (A2)+,D1        fetch cls-ptr from s
                ANDI.L  #$FFFFFFFD,D1   eliminate closure tag
                MOVE.L  D1,A0
                MOVE.L  (A0)+,A1        fetch p_fun
                MOVE.L  4(A1),D1        fetch arity behind ARGS op
                CMP.L   D1,D0           check arity conformity
                BNE     ERROR
                MOVE.L  A0,-(A2)        push p_env to s
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                MOVE.L  A1,A4           set P_C to fun-ptr
                MOVEM.L (SP)+,A0-A1
                RTS

I_PLUS:         MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                ADD.L   D1,D0
                EORI.L  #3,D0           correct tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_MINUS:        MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                SUB.L   D1,D0
                ORI.L   #1,D0           correct tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_MULT:         MOVE.L  (A2)+,D0        fetch first operand from s
                ASR.L   #2,D0           eliminate tag
                MOVE.L  (A2)+,D1        fetch second operand from s
                ASR.L   #2,D1           eliminate tag
                MULS.W  D1,D0
                LSL.L   #2,D0
                ORI.L   #1,D0           set tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_EQ:           MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                CMP.L   D0,D1
                BEQ     I_EQ_TRUE
                MOVE.L  #7,-(A2)        push false
```

```
                RTS
I_EQ_TRUE:      MOVE.L  #3,-(A2)        push true
                RTS

I_GT:           MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                CMP.L   D1,D0
                BGT     I_GT_TRUE
                MOVE.L  #7,-(A2)        push false
                RTS
I_GT_TRUE:      MOVE.L  #3,-(A2)        push true
                RTS

I_ARGS:         MOVE.L  (A4)+,D0        fetch parameter n and ignore it (-8
                RTS

I_LINK:         MOVE.L  (A4)+,D0        fetch parameter n
                ADDI.L  #1,D0           increment link parameter
                MOVE.L  A3,-(SP)
                MOVE.L  D0,-(SP)
                JSR     DEREF           get (n+1)-th indirection starting at P_E
                MOVE.L  (SP)+,-(A2)     push it on s
                RTS

I_CREATE_CLOS:  MOVE.L  A6,-(A2)        push closure ptr on s
                ORI.L   #2,(A2)         tag closure
                MOVE.L  (A4)+,D0        fetch parameter n
                MOVE.L  (A4)+,(A6)+     fetch parameter p_fun, push it onto heap
                ADDI.L  #1,D0           increment link parameter

                MOVE.L  A6,-(SP)        first arg for COPY_ENV

                MOVE.L  A3,-(SP)
                MOVE.L  D0,-(SP)
                JSR     DEREF           get n-th indirection starting at P_E
                                        * which is the second arg of COPY_ENV

                JSR     COPY_ENV        copy env into heap
                MOVE.L  (SP)+,A6        fetch new heap ptr (res of COPY_ENV)
                RTS

I_DONE:         RTS                     no operation

ERROR:          BREAK


*******************************************************************************

                INCLUDE io-pack.ss
```